Analiza oraz wizualizacja danych meteorologicznych dla wybranych ośrodków narciarskich

Michał Bugno **Antek Piechnik**
prowadzący: **dr inż. Robert Marcjan**

2 marca 2009

Spis treści

1 Wstęp

1.1 Wizja

Głównym zadaniem projektu jest poznanie struktury danych typu GIS (geographical information system), jak również analiza oraz wykorzystanie tego typu danych w wizualizacji danych meteorologicznych. System docelowo ma za zadanie przedstawienie sytuacji meteorologicznej na podstawie danych zbieranych na bieżąco jak również danych historycznych zgromadzonych poprzednio. System ma również mieć możliwość udostępniania danych/wizualizacji historycznych na życzenie użytkownika. Do celów badania wydajności systemu wykorzystywane będą dane z przynajmniej dwóch źródeł informacji meteorologicznej, podczas gdy system ma domyślnie obsługiwać 4-5 stacji narciarskich (po kilka punktów na każdą stację).

1.2 Ocena ryzyka

Technologia Django (w konsekwencji również Python) daje dobre perspektywy rozwoju: Python jest językiem bogatym w biblioteki (m.in. do Oracle) i jako język dynamiczny daje możliwość łatwego rozszerzania aplikacji. Dobrze rokuje także projekt GeoDjango (http://geodjango.org/) w związku z czym można sądzić, że nie napotkamy na większe problemy implementacyjne (związane z technologią).

2 Ogólna struktura systemu

2.1 Baza danych

Wybraną bazą danych jest Oracle. Wyboru dokonaliśmy głównie ze względu na możliwość dokładnego poznania tego produktu w ramach projektu jak również ze względu na obszerne wsparcie dla danych GIS - Oracle Spatial.

2.2 Technologia

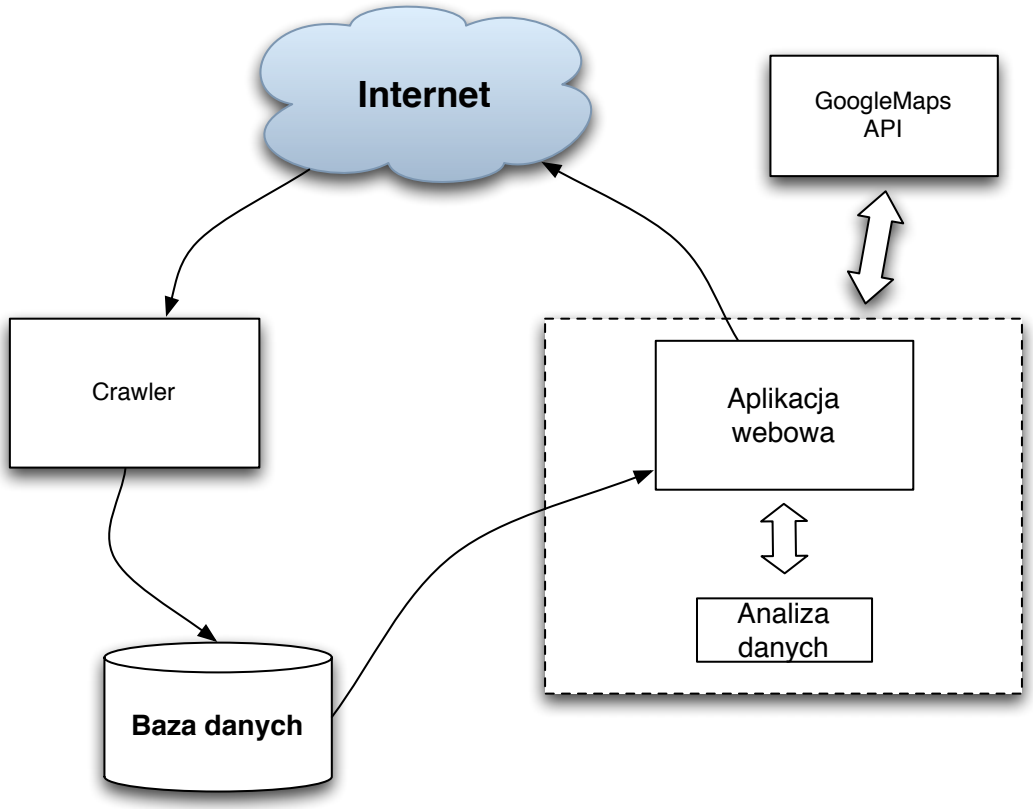
System jest tworzony w technologii Python Django – posiada ona wsparcie dla baz Oracle (w tym Spatial) oraz jest prosta i przejrzysta zapewniając bardziej elastyczny rozwój.

2.3 Crawler

Aplikacja jest w rzeczywistości skryptem mającym na celu pobranie odpowiednich danych z wcześniej przygotowanych źródeł (stron internetowych udostępniających informacje meteorologiczne dla konkretnych ośrodków). Będzie on miał również możliwość aktualizowania bazy danych o pobrane informacje, po uprzednich skonwertowaniu ich do odpowiedniego formatu.

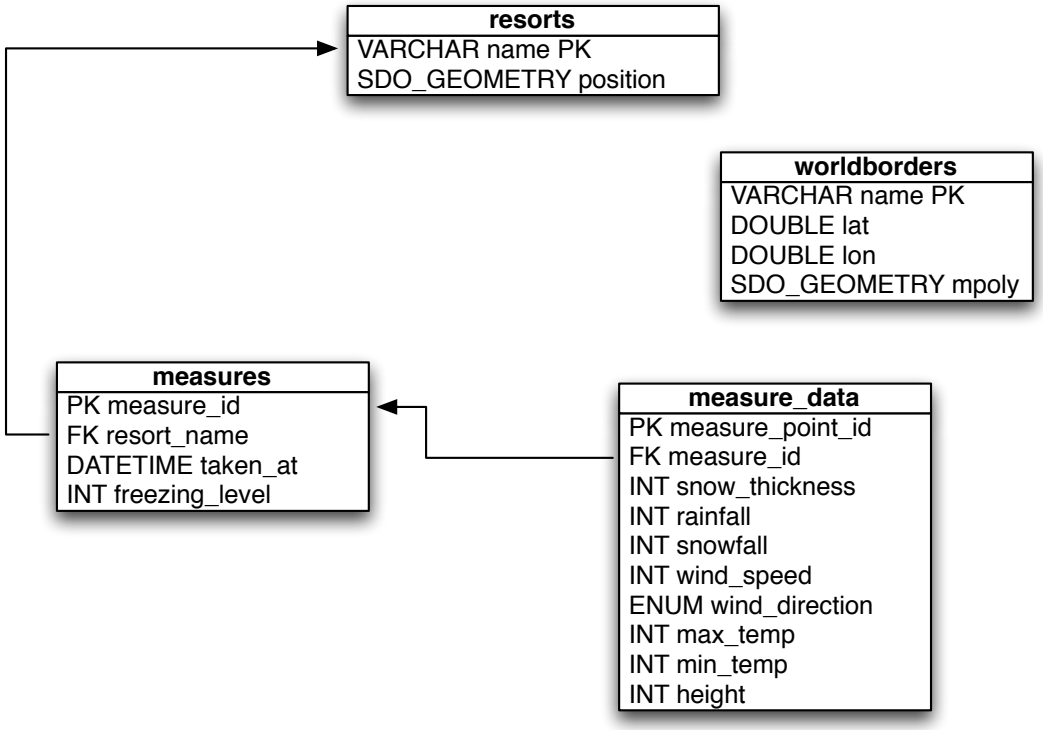
2.4 Wizualizacja

Wizualizacja zostanie utworzona w oparciu o dane wygenerowane przez kontroler analizy danych oraz o API systemu Google Maps który pozwoli na estetyczną wizualizację osiągniętych wyników analizy.



2.5 Entity Relationship Diagram

Poniższy model danych nie jest dokładny, to znaczy tabele `resorts` oraz `worldborders` są już stworzone natomiast pomiary zostaną dodane dopiero po stworzeniu crawlera w związku z czym model może nie wyglądać tak samo.



3 Aplikacja Django

3.1 Modele

Zdefiniowane w aplikacji modele to:

WorldBorders odpowiada za przechowywanie granic państw. Pola modelu:

name nazwa państwa

lat szerokość geograficzna

lon długość geograficzna

mpoly typ `SDO_GEOMETRY` przechowywujący `MultiPolygon` który definiuje granice państwa

Resorts przechowuje ośrodki dla których posiadamy dane pogodowe. Pola:

name nazwa miasta

position dwuwymiarowa geometria `Point` przechowywująca długość i szerokość geograficzną miasta

3.2 Metody

3.2.1 Wybranie obiektu z bazy

```
austria = WorldBorders.objects.filter(pk=Austria)[0]
first_resort = Resorts.objects.all()[0]
```

3.2.2 Państwo, w którym leży resort

```
country = first_resort.country()
print country.name # => ‘‘Austria‘‘
```

3.2.3 Najbliższe resorty

```
resorts = first_resort.within_distance(30)
# => resorty w odległości 30km od danego (bez danego)
```

3.3 SQL

3.3.1 Stworzenie tabel

Kluczowy dla projektu jest oczywiście typ `SDO_GEOMETRY` który umożliwia wykonywanie specyficznych zapytań geograficznych. Reszta pól to dodatkowe informacje na temat państwa/miasta.

```
CREATE TABLE "WORLD_WORLDBORDERS" (
    "NAME" NVARCHAR2(50) NOT NULL PRIMARY KEY,
    "LAT" DOUBLE PRECISION NOT NULL,
    "LON" DOUBLE PRECISION NOT NULL,
    "MPOLY" MDSYS.SDO_GEOMETRY NOT NULL
);
```

```
CREATE TABLE "WORLD_RESORTS" (
    "NAME" NVARCHAR2(50) NOT NULL PRIMARY KEY,
    "POSITION" MDSYS.SDO_GEOMETRY NOT NULL
);
```

Tabela pomiarów posiada dodatkowo klucz obcy do ośrodków aby pomiar można było zaklasyfikować do danego ośrodka.

```
CREATE TABLE "WORLD_MEASURES" (
    "ID" NUMBER(11) NOT NULL PRIMARY KEY,
    "RESORT_ID" NUMBER(11) NOT NULL REFERENCES "WORLD_RESORTS" ("ID")
         DEFERRABLE INITIALLY DEFERRED,
    "TEMP" NUMBER(11) NOT NULL,
    "TAKEN_AT" DATE NOT NULL
);
```

Ustawienia metryki Aby Oracle wiedział, jak wygląda metryka, należy poinformować go ustalając odpowiednie wartości graniczne oraz dokładność dla kolumn Spatial. W tym wypadku informujemy, że kolumna `MPOLY` tabeli `WORLD_WORLDBORDERS` posiada zakres długości od -180 do 180 oraz szerokości od -90 do 90 z dokładnością co 0.05.

```
INSERT INTO USER_SDO_GEOM_METADATA
   ("TABLE_NAME", "COLUMN_NAME", "DIMINFO", "SRID")
   VALUES (
    'world_worldborders',
    'mpoly',
    MDSYS.SDO_DIM_ARRAY(
      MDSYS.SDO_DIM_ELEMENT('LONG', -180.0, 180.0, 0.05),
      MDSYS.SDO_DIM_ELEMENT('LAT', -90.0, 90.0, 0.05)
    ),
    4326
  );
```

Indeksy Aby zapytania mogły funkcjonować należy stworzyć indeksy na kolumnach spatial. Służy do tego celu polecenie

```
CREATE INDEX "WORLD_RESORTS_POSITION_ID"
ON "WORLD_RESORTS"("POSITION")
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Sekwencje Warto wspomnieć, że część tabeli ma klucze główne liczbowe i aby nie przejmować się ich numerowaniem stworzyć należy sekwencje. Do tego celu użyliśmy: `CREATE SEQUENCE WORLD_WORLDBORDERS_SQ` i analogiczne dla pozostałych tabel.

3.3.2 Spatial queries

Ośrodki w pobliżu danego ośrodka

```
SELECT "WORLD_RESORTS"."ID", "WORLD_RESORTS"."NAME",
       SDO_UTIL.TO_WKTGEOMETRY("WORLD_RESORTS"."POSITION")
FROM "WORLD_RESORTS"
WHERE
    (SDO_WITHIN_DISTANCE("WORLD_RESORTS"."POSITION",
        SDO_GEOMETRY(POINT (10.7498 46.96297), 4326),
        \'distance=20000.0\') = \'TRUE\'
    AND NOT ("WORLD_RESORTS"."ID" = 832))
```

`SDO_WITHIN_DISTANCE` to funkcja sprawdzająca czy geometria z pierwszego argumentu znajduje się w pewnej odległości od geometrii drugiego. Jak widać drugą geometrię tworzymy przedstawiając dane geograficzne ośrodka w postaci Well-Known Text: `POINT(10.749, 46.962)` pierwsza jest natomiast do tej postaci konwertowana przez funkcję `TO_WKTGEOMETRY`. Trzeci argument to odległość jako liczba metrów. Cała funkcja zwraca true gdy warunek spełniony.

Państwo, w którym znajduje się ośrodek

```
SELECT "WORLD_WORLDBORDERS"."ID", "WORLD_WORLDBORDERS"."NAME",
       "WORLD_WORLDBORDERS"."LAT", "WORLD_WORLDBORDERS"."LON",
       SDO_UTIL.TO_WKTGEOMETRY("WORLD_WORLDBORDERS"."MPOLY")
FROM "WORLD_WORLDBORDERS"
WHERE SDO_CONTAINS("WORLD_WORLDBORDERS"."MPOLY",
    SDO_GEOMETRY(POINT (10.7498 46.96297), 4326)) = \'TRUE\'
```

W tym wypadku używamy funkcji `SDO_CONTAINS` która zwraca prawdę, gdy druga gemoetria całkowicie zawiera pierwszą. W naszym przypadku pierwszą geometrią jest wielobok przedstawiający granice państwa w tabeli państw natomiast druga to punkt reprezentujący ośrodek. W ten sposób zwracamy wszystkie państwa których granice obejmująten punkt (w większości przypadków będzie to jeden rekord).

3.4 Integracja z GoogleMaps API

API GoogleMaps jest w języku JavaScript. Wszystkie funkcje, ktorych używamy są zdefiniowane w widokach znajdujących się w katalogu `world/templates`.

Głównym widokiem jest `layout.html` i on definiuje mapę i podstawowe funkcje wyświetlania resortów. Należy pamiętać, że GoogleMaps działają tylko dla domeny, która została podana przy generowania klucza, dlatego jeśli domena będzie inna (aktualnie `http://localhost:8000`) należy wygenerować nowy klucz i zmienić widok `world/templates/layout.html`.

3.4.1 Granice Austrii

Granice wyświetlane są przez API KML-owe GoogleMaps. Reprezentację KML granic państwa mozna łatwo wyciągnąć z bazy danych, jednak API nie pozwala na dynamiczne wyświetlanie plików KML i muszą one być dostępne publicznie w Internecie. Z tego powodu wproowadzona jest redundancja danych: w pliku `world/templates/layout.html` znajduje się linia

```
var kml = new GGeoXml("http://student.agh.edu.pl/msq/austria.kml");
```

która definiuje położenie pliku do wyświetlania. W razie potrzeby taki plik można wygenerować za pomocą następujących poleceń:

```
python manage.py shell
> from world.models import WorldBorders
> austria = WorldBorders.objects.filter(pk="Austria")[0]
> print austria.mpoly.kml
```

3.5 Fikstury

Dane do aplikacji mogą zostać ponownie załadowane do bazy (włącznie z uprzednim jej wyczyszczeniem). W plikach `worldborders.fixtures` i `resorts.fixtures` znajdują się odpowiednie dane w prostym formacie tekstowym. Aby załadować je do bazy należy wykonać następujące polecenia:

```
python manage.py shell
> from world import load
> load.load_fixtures()
> # lub load.load_worldborders()
> # lub load.load_resorts()
```

Metoda `load.load_fixtures()` czyści bazę i tworzy wszystkie fikstury natomiast metody `load.load_worldborders()` i `load.load_resorts()` działają na poszczególnych tabelach.

4 Crawler

Crawler jest napisany w języku Ruby i służy do pobierania danych ze strony http://www.snow-forecast.com. Docelowo będzie to prosty skrypt oparty o metodologię *Extract–Transform–Load*.

4.1 Extract

- uruchamiamy skrypt z parametrem adresu strony (w zasadzie chodzi o wybrany szczyt)
- skrypt analizuje stronę za pomocą parsera HTML+XML Nokogiri
- dane zapisywane są w prostej postaci w tablicy
- skrypt znajduje link do danych z poprzedniego okresu, odwiedza go i powtarza proces

Dane zapisywane są przez

```
file = File.open(filename, "w")
file.write(Marshal.dump(data))
```

Aby je odczytać wystarczy

```
data = Marshal.load(File.read(filename))
```

5 Plan prac

Plan jest ułożony malejąco względem priorytetów.

5.1 Crawler

- część *Transform* przerabiające surowe dane na potrzeby aplikacji
- część *Load* tworząca SQL-e i wykonująca je w kontekście bazy

5.2 Import danych

- utworzenie odpowiednich tabel do przetrzymywania danych, utworzenie relacji między tabelami
- zmapowanie tabel w modelach Django
- ustalenie formatu danych oraz ich konwersja do formatu odpowiadającego modelowi w bazie danych
- zaimportowanie danych do bazy danych
- utworzenie części aplikacji odpowiedzialnej za aktualizację zbieranych danych

5.3 Analiza danych

- umożliwienie wyświetlania prognozowanych danych dla resortów, dla których istnieją pobrane dane (podpierając się GIS)
- implementacja systemu 'przewidującego' pogodę na podstawie poprzednich danych (na życzenie lub w sytuacji gdy dane prognozowane nie są dostępne)
- porównywanie danych z różnych ośrodków oraz przedstawienie raportów
- dobieranie najoptymalniejszego resortu spośród dostępnych raportów na bazie preferencji użytkownika
- możliwość zacieśnienia przeszukiwanych/porównywanych resortów tylko i wyłącznie do określonego regionu
- dobór ośrodka na podstawie aktualnej pozycji użytkownika (np. odległość od domu/odległość od ośrodka w którym użytkownik znajduje się aktualnie, a np. nie jest zadowolony)

5.4 Wizualizacja danych

- prezentowanie aktualnych danych w postaci różnego rodzaju wykresów (np. stosunek świeżego śniegu do grubości pokrywy śnieżnej, prezentacja temperatur na różnych wysokościach)
- generowanie wykresów na podstawie danych historycznych (np. temperatury z ostatniego tygodnia)

- wizualizacja porównawcza różnych ośrodków (prezentacja różnic w temperaturze, wietrze i śniegu dla dwóch lub więcej ośrodków)
- przedyskutowanie dalszych możliwości wizualizacji danych
- wizualizacja danych prognozowanych za pomocą Google Maps API

5.5 Dodatkowe

- wyszukanie serwisów prognozujących pogodę dla konkretnych regionów w Austrii (dostosowanie crawlera)
- implementacja wyszukiwania ośrodków
- utworzenie możliwości deklarowania preferencji przez użytkownika oraz ich zapisywanie
- możliwośc wyszukiwania bazującego na preferencjach użytkownika, poszerzenie systemu wyszukiwania
- próba wykorzystania Google Maps API do prezentowania dokładniejszej odległości danego ośrodka od użytkownika (na podstawie jego aktualnej pozycji np. w preferencjach)

6 Linki

Źródła (repozytorium Git) http://github.com/michalbugno/projekt-oszbd/

Projekt Django http://www.djangoproject.com/

Projekt GeoDjango http://geodjango.org/

API GoogleMaps http://code.google.com/apis/maps/documentation/

System kontroli wersji Git http://git-scm.com/

Python http://www.python.org/

Ruby http://www.ruby-lang.org/